

1 Attorney Docket No. 82937

2
3 FUNCTIONAL ELEMENT TEST TOOL AND METHOD

4
5 STATEMENT OF THE GOVERNMENT INTEREST

6 The invention described herein may be manufactured and used
7 by or for the Government of the United States of America for
8 Governmental purposes without the payment of any royalties
9 thereon or therefore.
10

11 CROSS-REFERENCE TO RELATED PATENT APPLICATIONS

12 Not applicable.
13
14

15 BACKGROUND OF THE INVENTION

16 (1) Field of the Invention

17 The present invention relates generally to software testing
18 and, more specifically, to a software test tool operable for
19 providing an independent test environment for testing computer
20 program system functional elements which may have multiple
Application Data Exchange (ADE) interfaces.

1 Description of the Prior Art

2 Large scale hardware and software systems and application
3 may typically include many different functional elements or
4 modules. In many cases, the various functional elements or
5 modules are developed concurrently. As used herein, a
6 functional element is a software module which performs a unique
7 software task and which may have multiple interfaces with other
8 functional elements and/or with an application comprised of
9 numerous functional elements and/or with an overall system
10 comprised of a plurality of applications. In a preferred
11 embodiment of the present invention, the functional element
12 performs one or more tasks which utilize an inter-task interface
13 or module-to-module communication protocol or mechanism. Each
14 functional element may have multiple interfaces. The interface
15 sets forth constraints on formats, timing, and/or other factors
16 required by an interaction of functional elements that perform
17 different tasks within a computer system.

18 Once the system is assembled, various problems may occur
19 that cannot be easily traced to a particular functional element
20 or module. In many cases, the problems relate to errors that
21 occur at the functional element or module interfaces. As an
22 example, one module or functional element might test and analyze

1 different underwater acoustic propagation signal propagation
2 loss models. If the overall system response is not as expected,
3 it may be difficult to determine whether the particular
4 functional element for analyzing signal propagation loss is
5 operating correctly or whether the fault lies elsewhere.
6 Moreover, it may typically be difficult to monitor software
7 interface operation of any particular functional element with
8 respect to other functional elements and the overall system.

9 Where systems involve complex meshings of inter-process
10 communications, a known technique for implementing the
11 communication of task messages has been to use a shared memory
12 for passing data between tasks. However, with this technique
13 the problem of determining the fault which produces a given
14 error is even greater.

15 Consequently, there remains a long felt but unsolved need
16 for improved testing of functional elements to ensure that
17 accurate processing occurs within each functional element and
18 that communications between the functional elements are
19 consistent with the interface protocols. Those skilled in the
20 art will appreciate the present invention that addresses the
21 above and other problems.

1 SUMMARY OF THE INVENTION

2 Accordingly, it is an object of the present invention to
3 provide an improved system and method for testing functional
4 elements of a computer software system.

5 Another object is to provide a system and method as
6 aforesaid which provides a controlled test environment that can
7 facilitate off-line quantitative analysis of the test results
8 and collected data.

9 A further object is to provide a system and method as
10 aforesaid which is a stand alone functional test tool that is
11 much easier and faster to operate than a debugger or hand
12 inspection of the application output results.

13 A still further object is to provide a system and method as
14 aforesaid which enables software to reduce the overall number of
15 defects that occur during the development phase.

16 A yet further object is to provide a system and method as
17 aforesaid which is of special utility in application systems in
18 which inter-process communication is implemented by techniques
19 of using a shared memory scheme in order to pass data between
20 tasks.

21 These and other objects, features, and advantages of the
22 present invention will become apparent from the drawings, the

1 descriptions given herein, and the appended claims. However, it
2 will be understood that above listed objects and advantages of
3 the invention are intended only as an aid in understanding
4 aspects of the invention, are not intended to limit the
5 invention in any way, and do not form a comprehensive list of
6 objects, features, and advantages.

7 In accordance with the present invention, a method is
8 disclosed for a stand-alone testing environment for a functional
9 element of an application. The application may be a subpart of
10 an overall system. The functional element may have a plurality
11 of interfaces with the application and/or overall system. The
12 method may comprise one or more steps such as, for instance,
13 providing a computerized dialog to permit a user to create an
14 input data file for the functional element, prompting a user for
15 functional element interface tasks that have been previously
16 developed utilizing the stand-alone testing environment.

17 Alternatively, the dialog permits a user to start the functional
18 interface tasks and a user supplied application system task.

19 The plurality of interfaces are monitored.

20 Other steps may include storing a unique interface file for
21 the functional element interface tasks and/or displaying a
22 status window while running the functional interface tasks. In

1 a preferred embodiment, the step of creating a test case
2 generation file provides the user with task creation options
3 related to the functional interface tasks. Other steps may
4 include compiling the test case generation file to create a test
5 case executable file for producing the functional interface
6 tasks and/or storing the input data file in a user defined file
7 such that the user defined file may be viewed and edited outside
8 of the stand alone testing environment.

9 An embodiment of the present invention provides a method
10 for testing a functional element of a computer system with a
11 stand-alone functional element test tool wherein the functional
12 element has at least one interface for communicating with other
13 functional elements of the computer system using an interface
14 protocol. In this embodiment, the method comprises one or more
15 steps such as, for instance, creating an input data file for the
16 functional element by prompting a user for data format and
17 content, storing the input data file, creating a test generation
18 file by providing the user with a plurality of task creation
19 options such that the selected task creation options are input
20 into the test generation file, compiling the test generation
21 file to produce a test case executable file in a predetermined
22 inter-process communication protocol based on the selected task

1 creation options, initiating a test utilizing the test case
2 executable file and the input data file for testing the
3 functional element and the interface(s) by monitoring a status
4 of the test, and storing test result data related to the test.

5 The step of creating a test generation file may further
6 comprise selecting test initiation features and/or providing at
7 least one user defined button wherein the user defined button is
8 user operable for the step of initiating the test. The method
9 may further comprise playing back the test result data and/or
10 providing a file viewer for the input data. Thus, the method
11 permits comparing the test result data with expected results
12 from the functional element utilizing the input data file.

13 Thus, the present invention also provides a system
14 operative for testing performance validity and accuracy of a
15 first computer program functional element which may comprise one
16 or more elements such as, for instance, a test case data file
17 producing subsystem for facilitating the production by a user of
18 at least one file of test case data. The test case data
19 producing subsystem can then be used for identification of an
20 input data structure in order to prompt a user for input values
21 of the test case data. Preferably, the test case data producing
22 subsystem is operative to store one or more files of test case

1 data. Other elements may include a test case generation file
2 producing subsystem for facilitating the production by a user of
3 a test case generation file. The test case generation file
4 producing subsystem provides a plurality of user options to
5 facilitate the user in testing operation of the first computer
6 program functional element and at least one communication
7 interface. Moreover, the system may comprise a test case
8 execution subsystem to operate the first functional element
9 based on the selected user options and the test case data. The
10 test case execution subsystem preferably provides a monitor for
11 operation of the first functional element. Preferably the test
12 case execution subsystem is operable for executing operation of
13 a compiled executable file produced from the test case
14 generation file. The test case execution subsystem may
15 preferably be operable to effect operation of a second
16 functional element simultaneously with operation of the first
17 functional element such that the test case execution subsystem
18 is operable to monitor the interface between the first function
19 element and the second functional element. The test case
20 execution producing subsystem is also preferably operative to
21 tag an output of the first functional element with an indication
22 of the task status. The test case generation file producing

1 subsystem is also preferably operative to facilitate the user
2 selecting a predetermined test initiation event to start flow of
3 the test case data into the first functional element.

5 BRIEF DESCRIPTION OF THE DRAWINGS

6 A more complete understanding of the invention and many of
7 the attendant advantages thereto will be readily appreciated as
8 the same becomes better understood by reference to the following
9 detailed description when considered in conjunction with the
10 accompanying drawings wherein corresponding reference characters
11 indicate corresponding parts throughout several views of the
12 drawings and wherein:

13 FIG. 1 is a schematic which indicates the general
14 functioning of a stand-alone test tool in accord with the
15 present invention;

16 FIG. 2 is a schematic of an embodiment of a stand-alone
17 test tool as the tool may be configured to cooperate with a
18 given computer application system; and

19 FIG. 3 is a block diagram indicating a flow and structure
20 of a stand-alone test tool in accord with the present invention.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

Referring to FIG. 1, in accord with the present invention, a stand-alone functional element (SAFE) test tool 10 has been developed to provide an independent test environment for one or more functional elements 18, which are desired to be the object of testing, within a given application or system. In a preferred embodiment, the SAFE test tool can be utilized to test any test object software which utilizes an inter-module communication mechanism or interface protocol.

An illustrative embodiment of the inter-process communication protocol is the Application Data Exchange (ADE) protocol, which has been custom developed for the Navy for use in connection with software application programs. Illustrative of one such Navy application program is a Sensor Performance Prediction Functional Segment (SPPFS) large-scale software module. SPPFS has multiple interfaces with software programs of an integrated system of ASW equipment digital processors. Briefly, SPPFS performs casting and analysis of different acoustic signal propagation loss models used in ASW combat control. More particularly, it functions in connection with sonar-based antisubmarine warfare equipment known as system AN/SQQ-89(V) 15. However, both the SPPFS and the AN/SQQ-89(V) 15

1 are referred to for purposes of illustrating an environment, in
2 which the present invention is utilized, and in and of
3 themselves form no part of the invention.

4 ADE uses queues to communicate between application tasks.
5 The concept of queue within ADE can be viewed as a mechanism for
6 inter-process signaling. Each queue consists of one or more
7 messages. Messages in turn contain information. An example of
8 a queue with an implementation in the environment of ASW systems
9 is the queue which is used by a propagation loss analysis
10 function task element of a sonar signal analysis computer
11 program to send equipment settings to the program's interface
12 function task elements. This provides communication between the
13 two tasks.

14 ADE uses a shared memory scheme in order to pass data
15 between tasks. ADE manages the shared memory. Therefore when
16 employed with SPPFS this functionality is invisible to the
17 implementation of the SPPFS program task. However, the memory
18 requirements of the implementation must be known and allocated
19 via ADE functions (e.g., AdeAlloc). Message passing with ADE
20 can be viewed as the notification by an ADE protocol message to
21 a task that information (sometimes in this specification and its

1 appended claims simply referred to as "data") is ready and that
2 the data is located at a specific shared memory location.

3 ADE uses a hierarchical structure for storing information
4 within the ADE database. There are four levels within the
5 hierarchy, "Queue", "Application", "Topic", and "Item", where
6 Queue is the most general and Item is the most specific. A Queue
7 represents a highest level within the hierarchy. Within the
8 SPPFS application program implementation, the Queue is typically
9 the interface with the high level task. The Application is the
10 second level and is associated with a queue. The functionality
11 of an Application is typically more specific than that of a
12 queue. However, within the SPPFS application the two often
13 encompass the same functionality. The third level is a Topic
14 which is associated with an Application. Again, the Topic is
15 more specific than the queue. Finally the fourth level is the
16 Item which is based on a Topic.

17 The SAFE test tool provides an excellent test bed for
18 testing functional element changes as well as software drops
19 because the user has the capability of generating, collecting,
20 and responding to all interfaces of the specified system
21 functional element. Although the SAFE test tool prompts the
22 user for input to perform testing, it is assumed that the user

1 understands the test object functional element operation within
2 the overall system as well as the inter-module communication or
3 interface protocol, e.g., the ADE protocol.

4 Referring again to FIG. 1, there is schematically shown
5 SAFE test tool 10 interconnected with any number of interfaces,
6 as indicated by 12, 14, and 16, of test object functional
7 element 18 to provide an independent test environment. Each of
8 interfaces 12, 14, and 16 may have several modes of operation
9 which may be tested in accord with the present invention. SAFE
10 test tool 10 provides a stand-alone test environment that may be
11 utilized to simulate the application or overall system 19 with
12 respect to test object functional element 18 and monitor any
13 number of interfaces for proper operation, such as element
14 interfaces 12, 14, and 16. The SAFE test tool 10 and the
15 application or system 19 each communicate via a suitable linking
16 arrangement, such as computer bus 19-1, with the above-discussed
17 shared memory 19-2, that is involved in the shared memory scheme
18 to pass data between tasks. Also per this earlier discussion,
19 shared memory 19-2 has four levels of hierarchical structure:
20 Queue 19-2A, Application 19-2B, Topic 19-2C, and Item 19-2D.

21 SAFE test tool 10 can also be utilized in other
22 configurations. For instance, in FIG. 2, SAFE test tool 10 may

1 be connected to one or more test object functional elements,
2 such as functional elements 1, functional element 2, functional
3 element 3, and functional element N, designated as 20, 22, 23,
4 and 24, respectively, wherein communication paths 26, 28, 29,
5 and 30, may comprise one or more ADE interfaces. Each ADE
6 interface may have several modes of operation. In this
7 embodiment, the test object functional elements may be part of
8 or form a software application program such as, for example, one
9 of the functional elements of the above described Sensor
10 Performance Prediction Functional Segment (SPPFS) 31. In turn,
11 SPPFS is part of an overall large-scale integrated system, such
12 as the above referred to AN/SQQ-89(V)15 combat control system
13 (not shown). The functional elements of SPPFS interface with
14 various application programs of the large scale integrated
15 system through multiple interfaces, with interface tasks
16 constituting inter-process communications. There may be other
17 elements such as a CORBA (common object request broker
18 architecture) link 31-1 and a CORBA software bridge 32 which may
19 be utilized with many functional elements utilizing, preferably,
20 ADE interfaces. In another arrangement, AN/SQQ-89(V)15 External
21 Interfaces 34 as indicated in dashed lines, may operate through

1 link 31-1 and CORBA bridge 32 for testing one or more functional
2 elements 20-24 and the various associated interfaces.

3 FIG. 3 provides an overview of the details of operation of
4 SAFE test tool 10. In order to effectively interface with a
5 test object functional element, i.e., the functional element
6 involved in the run test case step 86 of a test. SAFE test tool
7 10 preferably operates in several selectable modes of operation
8 or, in other words, is comprised of several selectable
9 subsystems, as indicated at select subsystem 50. These
10 subsystems of SAFE test tool 10 permit the user to create input
11 data 52, create an interface test 54, application, and to run
12 the tests 56.

13 Input file creator mode or subsystem 52 of SAFE test tool
14 10 provides a mechanism that allows a user to create one or more
15 input data files that can eventually be accessed during test run
16 time. During input file creation, SAFE test tool 10 provides a
17 dialog with the user as indicated at 58 and 60. SAFE test tool
18 10 allows the user to identify the structure of the input data
19 and then uses this structure to prompt the user as to possible
20 input values 60 in conjunction with user dialog 58. This is
21 accomplished by prompting the user for data format and content.
22 The format may be manually entered or the format may be

1 automatically determined. For instance, utilizing a drop-down
2 form or other selection means, the user may identify the data
3 structure to be utilized. Once the data format is identified,
4 the data format is then utilized by SAFE test tool 10 to prompt
5 the user to enter input values. When completed, the data is
6 stored in a user-defined data input file 62. User-defined data
7 input file 62 can now be used as an input file for future test
8 cases. Preferably, the format of user-defined data input file
9 62 is such that the file contents may be viewed and modified
10 outside of the environment of SAFE test tool 10 for ease of use.
11 For instance, the file may be readable utilizing file editor 64
12 which may be a text file editor or any other suitable editor
13 which preferably does not require the use of SAFE test tool 10.

14 Create interface test case mode or subsystem 54 of SAFE 10
15 provides a mechanism that allows the user to create a test case
16 generation file. In one illustrative embodiment, the test
17 generation file is recorded in the form of a high-level
18 interface language, custom developed for ADE. The development
19 of such a custom language is within the routine skill of
20 programmers who develop special purpose compilers. More
21 generally, the high-level interface language can be a
22 development for any other interface protocol where such other

1 interface protocol is involved. In this embodiment, it is
2 assumed that the user is knowledgeable of the functional element
3 interface task to be tested. The file generation is
4 accomplished by providing the user with interface task creation
5 options 66 and user input 68 and may include options such as
6 Create Queue, Edit Queue, Begin Enumeration, and the like.

7 This description now proceeds to details of various
8 preferred functional element task creation options. Create
9 Queue allows SAFE test tool 10 to be able to create a Queue
10 component of an ADE message which represents a function or
11 functions to be performed for a test case. This component
12 comprises the information (herein and the appended claims
13 sometimes simply referred to as "data") stored in the Queue
14 level of the ADE hierarchical shared memory. For each Queue
15 message component, SAFE test tool 10 allows the creation of a
16 user-defined Queue message component processing procedure or the
17 use of a default Queue message component processing procedure.
18 SAFE test tool 10 preferably has an edit queue option whereby
19 SAFE test tool 10 allows editing of each queue. However, in a
20 one preferred embodiment, once the Queue is created, SAFE test
21 tool 10 is not able to delete the Queue message component.
22 Another option is the Process Queue Messages option whereby SAFE

1 test tool 10 preferably is able to add messages that will be
2 automatically handled within the test case. For this purpose,
3 SAFE test tool 10 writes data to an output file specified by the
4 user. SAFE test tool 10 allows for the processing of multiple
5 message contents in the same Queue message component. SAFE test
6 tool 10 does not allow the addition of messages until the Queue
7 message component for the test case has been created. The
8 option is also provided for creating an Application component of
9 an ADE message comprising data stored in the Application level
10 of the shared memory, whereby SAFE test tool 10 preferably
11 permits creation of multiple Application message components for
12 each test queue. SAFE test tool 10 permits Application message
13 component creation only after a Queue message component has been
14 successfully created. An Edit Application option permits SAFE
15 test tool 10 to modify each Application message component. A
16 Create Topic message component option permits SAFE tool 10 to
17 similarly create multiple Topic message components for each test
18 application. Topic message components are capable of being
19 created only after an Application message component has been
20 successfully created. Edit Topic permits modification of each
21 Topic message component. Create/Edit Item allows SAFE test tool
22 10 to be able to create and/or edit multiple Items message

1 components. Note that Items message components are created only
2 after a Topic message component has been successfully created.
3 Enumerate on Application, Topic, or Queue, allows SAFE test tool
4 10 to be able to begin an enumeration on an application, topic,
5 or item that has been defined in the task to be tested. SAFE
6 test tool 10 automatically saves the server, Queue message
7 component, and Topic message component identifiers upon
8 successful enumeration. In addition, SAFE test tool 10
9 preferably provides the capability of allocating memory,
10 requesting data, or requesting notice when the enumeration is
11 satisfied. An Allocate/Free Memory option permits SAFE test
12 tool 10 to be able to allocate and free memory and associate
13 each memory block action with a given action (e.g.,
14 enumeration). A Send/Receive Data option permits SAFE test tool
15 10 to send and receive data to the task being tested. An
16 Initiate Input Data option preferably permits SAFE test tool 10
17 to provide the capability of initiating input data by allowing
18 the user to create, for instance, buttons or other indicators.
19 Upon selection of the button, SAFE test tool 10 would then
20 initiate data to the interface and functional element being
21 tested. The input data will be accessed from a user defined
22 input data file. The option to Insert Header Files permits SAFE

1 test tool 10 to provide the user with the ability to insert
2 header files into the software as needed. This can preferably
3 be performed automatically or manually.

4 The selected options are input into a generation file that
5 is used to create the functional element test case. One of the
6 options, as discussed above, permits the user to accept/initiate
7 data from/to the function element for the interface (preferably
8 ADE) task being tested. Data may preferably be initiated
9 automatically upon the occurrence of some event (e.g., receipt
10 of a message) or manually via a user-defined button as discussed
11 above. Upon completion of the test case generation process,
12 test case generation file 70 is available. In a preferred
13 embodiment, test case generation file 70 is compiled outside of
14 SAFE test tool 10 utilizing, for instance, by a compiler 72
15 which compiles the various options selected to create test case
16 executable 74.

17 Test case execution 56 of SAFE test tool 10 provides a
18 mechanism to run multiple test case executables when testing a
19 functional element application task as indicated at 84 and 86.
20 In a preferred embodiment, this testing employs a predetermined
21 interface protocol for inter-process communications, e.g., the
22 ADE protocol. Test case executable files 74 are therefore the

1 compiled executable version of the test case generation file
2 created within the test case creation 54 mode. For each test
3 case, SAFE test tool 10 preferably provides a monitoring window,
4 as indicated at 78, that displays the current status of the
5 given task. During test case generation stage 54, preferably
6 print statements (for instance print statements) are
7 automatically inserted that generate or print the status in
8 monitoring window 78. In addition, the user define options
9 (e.g., data initiation buttons) are displayed as dictated by
10 each test case. The run test case step 86 is then performed
11 under initiation by a start options function 84, using as inputs
12 the user defined filed 62 and test case executable 74. Upon
13 completion of the task execution, the results are output to the
14 test case and may be stored in data storage 80 for future
15 analysis or playback such as indicated at 82. SAFE test tool 10
16 is operable to maintain a unique interface file for each
17 functional element task as prescribed by the user.

18 Reference is again made to FIG. 3, for an example of use of
19 SAFE test tool 10 in connection with the SPPFS software
20 application program (see reference character 31, FIG. 2) and the
21 operation of SPFFS in examining and analyzing different acoustic
22 signal propagation loss models. Further, in this example, it is

1 assumed that SAFE test tool 10 employs the ADE interface
2 protocol for inter-process communications. Both the interface
3 requirements and the specific input data requirements are well
4 defined within the SPPFS application. Varying input data can be
5 supplied and outputs can be collected to data files for off-line
6 quantitative analysis utilizing playback 82, e.g., pointwise
7 comparison with simulated/expected results. In addition,
8 comparison of the different acoustic signal propagation models
9 with different environments can be facilitated via SAFE test
10 tool 10. When using SAFE test tool 10 in this way, the user
11 would be prompted for the command line that initiates the
12 selected SPPFS functional element to be the object of testing by
13 SAFE test tool 10. In addition, the user would be prompted for
14 the interface tasks as developed with interface task creation
15 options 66. These tasks would be initiated via SAFE test tool
16 10 with test case executable 74. Upon successful initiation at
17 84, the user would have the ability to initiate inputs to the
18 functional element to be tested. The responses from the test
19 object functional element being tested during run test case step
20 86 would be maintained via SAFE test tool 10 for later
21 comparison or output validation.

1 In accordance with the present invention, any application
2 that utilizes a particular inter-module communication mechanism
3 or interface such as the ADE protocol, or other interface
4 protocols, will be significantly enhanced by the use an
5 embodiment of SAFE test tool 10. SAFE test tool 10 facilitates
6 functional level testing and evaluation of the application
7 system (in the example, SPPFS) functional elements independently
8 of the remainder of the system. In addition, SAFE test tool 10
9 provides a controlled test environment that can facilitate off-
10 line quantitative analysis of the test results and collected
11 data. Finally, SAFE test tool 10 provides a good environment
12 for verifying software changes and status at a functional
13 element level. This mechanism is therefore a valuable tool in
14 the validation of software performance and processing accuracy.
15 Quality software can be developed in less time and with fewer
16 overall defects due to use of SAFE test tool 10 during the
17 development phase. This affords significant cost savings both
18 during the initial development phase as well as overall life
19 cycle maintenance.

20 It will be appreciated by those skilled in the art that the
21 invention can be implemented using a suitable programmed general
22 purpose computer or special purpose hardware, with program

1 routines or logical circuit sets performing as processors. Such
2 routines or logical circuit sets may also be referred to as
3 processors or the like.

4 Therefore, it will be understood that many additional
5 changes in the details, materials, steps and arrangement of
6 parts, which have been herein described and illustrated in order
7 to explain the nature of the invention, may be made by those
8 skilled in the art within the principle and scope of the
9 invention as expressed in the appended claims.

10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100